Recap: MDPs

- **Markov decision processes:**
  - States $S$
  - Actions $A$
  - Transitions $P(s'|s,a)$ (or $T(s,a,s')$)
  - Rewards $R(s,a,s')$ (and discount $\gamma$)
  - Start state $s_0$

- **Quantities:**
  - Policy = map of states to actions
  - Episode = one run of an MDP
  - Utility = sum of discounted rewards
  - Values = expected future utility from a state
  - Q-Values = expected future utility from a q-state
Recap: Optimal Utilities

- The utility of a state $s$: $V^*(s) = \text{expected utility starting in } s \text{ and acting optimally}$
- The utility of a q-state $(s,a)$: $Q^*(s,a) = \text{expected utility starting in } s, \text{ taking action } a \text{ and thereafter acting optimally}$
- The optimal policy: $\pi^*(s) = \text{optimal action from state } s$

Recap: Bellman Equations

- Definition of utility leads to a simple one-step lookahead relationship amongst optimal utility values:
  
  $\text{Total optimal rewards} = \text{maximize over choice of (first action plus optimal future)}$

- Formally:
  
  $V^*(s) = \max_a Q^*(s,a)$
  
  $Q^*(s,a) = \sum_{s'} T(s,a,s') \left[ R(s,a,s') + \gamma V^*(s') \right]$
Value Estimates

- Calculate estimates $V_k^*(s)$
  - Not the optimal value of $s$!
  - The optimal value considering only next $k$ time steps ($k$ rewards)
  - As $k \to \infty$, it approaches the optimal value

- Almost solution: recursion (i.e. expectimax)
- Correct solution: dynamic programming

Value Iteration

- Idea:
  - Start with $V_i^*(s) = 0$, which we know is right (why?)
  - Given $V_i^*$, calculate the values for all states for depth $i+1$:

  $$V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]$$

  - Throw out old vector $V_i^*$
  - Repeat until convergence
  - This is called a value update or Bellman update

- Theorem: will converge to unique optimal values
  - Basic idea: approximations get refined towards optimal values
  - Policy may converge long before values do
Example: Bellman Updates

$$V_{i+1}(s) = \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]$$

$$V_2((3,3)) = \sum_{s'} T((3,3), \text{right}, s') \left[ R((3,3)) + 0.9 V_1(s') \right]$$

max happens for $a=\text{right}$, other actions not shown

\[
= 0.9 \left[ 0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0 \right]
\]

Example: Value Iteration

- Information propagates outward from terminal states and eventually all states have correct value estimates
Convergence*

- Define the max-norm: $\|U\| = \max_s |U(s)|$

- Theorem: For any two approximations $U$ and $V$

  \[ \|U^{t+1} - V^{t+1}\| \leq \gamma \|U^t - V^t\| \]

  - i.e. any distinct approximations must get closer to each other, so, in particular, any approximation must get closer to the true $U$ and value iteration converges to a unique, stable, optimal solution

- Theorem:

  \[ \|U^{t+1} - U^t\| < \epsilon, \Rightarrow \|U^{t+1} - U\| < 2\epsilon\gamma/(1 - \gamma) \]

  - i.e. once the change in our approximation is small, it must also be close to correct

Practice: Computing Actions

- Which action should we chose from state $s$:
  - Given optimal values $V$?
    \[
    \arg\max_a \sum_{s'} T(s, a, s') [R(s, a, s') + \gamma V^*(s')] \]
  - Given optimal q-values $Q$?
    \[
    \arg\max_a Q^*(s, a) \]

  - Lesson: actions are easier to select from Q’s!

[DEMO – MDP action selection]
Utilities for a Fixed Policy

- Another basic operation: compute the utility of a state \( s \) under a fixed (generally non-optimal) policy.

- Define the utility of a state \( s \), under a fixed policy \( \pi \):
  \[
  V^\pi(s) = \text{expected total discounted rewards (return) starting in } s \text{ and following } \pi
  \]

- Recursive relation (one-step look-ahead / Bellman equation):
  \[
  V^\pi(s) = \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi(s')]
  \]

Policy Evaluation

- How do we calculate the \( V \)'s for a fixed policy?

- Idea one: turn recursive equations into updates
  \[
  V^\pi_0(s) = 0
  \]
  \[
  V^\pi_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s')[R(s, \pi(s), s') + \gamma V^\pi_i(s')]
  \]

- Idea two: it's just a linear system, solve with Matlab (or whatever)
Policy Iteration

- **Alternative approach:**
  - **Step 1: Policy evaluation:** calculate utilities for some fixed policy (not optimal utilities!) until convergence
  - **Step 2: Policy improvement:** update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values
  - Repeat steps until policy converges

- **This is policy iteration**
  - It’s still optimal!
  - Can converge faster under some conditions

**Policy Iteration**

- **Policy evaluation:** with fixed current policy $\pi$, find values with simplified Bellman updates:
  - Iterate until values converge

$$V_{i+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') \left[ R(s, \pi(s), s') + \gamma V_i(s') \right]$$

- **Policy improvement:** with fixed utilities, find the best action according to one-step look-ahead

$$\pi_{k+1} = \arg \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right]$$
Comparison

- Both compute same thing (optimal values for all states)
- In value iteration:
  - Every pass (or "backup") updates both utilities (explicitly, based on current utilities) and policy (implicitly, based on current utilities)
  - Tracking the policy isn’t necessary; we take the max
    \[ V_{i+1}(s) \leftarrow \max_a \sum_{s'} T(s, a, s') \left[ R(s, a, s') + \gamma V_i(s') \right] \]
- In policy iteration:
  - Several passes to update utilities with fixed policy
  - After policy is evaluated, a new policy is chosen
- Both are dynamic programs for solving MDPs

Asynchronous Value Iteration*

- In value iteration, we update every state in each iteration
- Actually, any sequences of Bellman updates will converge if every state is visited infinitely often
- In fact, we can update the policy as seldom or often as we like, and we will still converge
- Idea: Update states whose value we expect to change: If \(|V_{i+1}(s) - V_i(s)|\) is large then update predecessors of s
Reinforcement Learning

- Reinforcement learning:
  - Still have an MDP:
    - A set of states $s \in S$
    - A set of actions (per state) $A$
    - A model $T(s,a,s')$
    - A reward function $R(s,a,s')$
  - Still looking for a policy $\pi(s)$

- New twist: don’t know $T$ or $R$
  - I.e. don’t know which states are good or what the actions do
  - Must actually try actions and states out to learn

Example: Animal Learning

- RL studied experimentally for more than 60 years in psychology
  - Rewards: food, pain, hunger, drugs, etc.
  - Mechanisms and sophistication debated

- Example: foraging
  - Bees learn near-optimal foraging plan in field of artificial flowers with controlled nectar supplies
  - Bees have a direct neural connection from nectar intake measurement to motor planning area
Example: Backgammon

- Reward only for win / loss in terminal states, zero otherwise
- TD-Gammon learns a function approximation to V(s) using a neural network
- Combined with depth 3 search, one of the top 3 players in the world
- You could imagine training Pacman this way…
- … but it’s tricky! (It’s also P3)

Passive Learning

- Simplified task
  - You don’t know the transitions T(s,a,s’)
  - You don’t know the rewards R(s,a,s’)
  - You are given a policy π(s)
  - Goal: learn the state values
  - … what policy evaluation did

- In this case:
  - Learner “along for the ride”
  - No choice about what actions to take
  - Just execute the policy and learn from experience
  - We’ll get to the active case soon
  - This is NOT offline planning! You actually take actions in the world and see what happens…
Example: Direct Estimation

- Episodes:
  - (1,1) up -1
  - (1,2) up -1
  - (1,2) up -1
  - (1,3) right -1
  - (1,3) right -1
  - (2,3) right -1
  - (3,3) right -1
  - (3,2) up -1
  - (3,3) right -1
  - (4,3) exit +100
  - (done)

\[ V(1,1) \sim (92 + -106) / 2 = -7 \]
\[ V(3,3) \sim (99 + 97 + -102) / 3 = 31.3 \]

Model-Based Learning

- Idea:
  - Learn the model empirically through experience
  - Solve for values as if the learned model were correct

- Simple empirical model learning
  - Count outcomes for each \( s,a \)
  - Normalize to give estimate of \( T(s,a,s') \)
  - Discover \( R(s,a,s') \) when we experience \( (s,a,s') \)

- Solving the MDP with the learned model
  - Iterative policy evaluation, for example

\[ V^\pi_{t+1}(s) \leftarrow \sum_{s'} T(s, \pi(s), s') [R(s, \pi(s), s') + \gamma V^\pi_t(s')] \]
Example: Model-Based Learning

- **Episodes:**

  1. (1,1) up -1
  2. (1,2) up -1
  3. (1,2) up -1
  4. (1,3) right -1
  5. (2,3) right -1
  6. (3,3) right -1
  7. (3,3) right -1
  8. (3,2) up -1
  9. (3,3) right -1
  10. (4,3) exit +100
  11. (done)

  \[ T(<3,3>, \text{right, } <4,3>) = \frac{1}{3} \]

  \[ T(<2,3>, \text{right, } <3,3>) = \frac{2}{2} \]